الدوال (Functions) وكل مايتعلق بها في ++C

والـ : Function هي دالة لها تقوم بعمليه أو مجموعة عمليات ، سوء عمليات إدخال أو إخراج أو عمليات حسابية أو منطقية ، وتحتل الدالة موقعاً من البرنامج أي إنها جزء منه ، أو ممكن القول أن برنامج ++C يتكون من مجموعة من الدوال
أي لها وضيفة معينة أو عدد وظائف و تنتهي function وتموت عندا نهاية عملها
وال main تدعى أيضاًfunction

ومن فوائدها- :

-1تساعد في اختصار كتابه كود للبرنامج إذا يكتفي باستعادتها باسمها فقط لتقوم بالعمل المطلوب.

-2تلافي عملية التكرار في خطوات البرنامج التي تطلب عملاً طويلاً وشاقاً.

-3تساعد الدوال في عملية البرمجة نفسها.

-4توفر مساحة من الذاكرة المطلوبة للبرنامج.

-5اختصار عملية زمن البرمجة وتنفيذ البرنامج بأسرع وقت ممكن.

-6تسهل كثيراً في مراجعة وتصحيح الكود والتعديل فيه ، إي يتم تقسيم البرنامج إلى عدد function لكل داله لها عمل معين فعندما نريد أن نعدل في جزء معين في الكود نذهب إلي الـ function التي تحوي الكود الذي نريده بدلاً من ملاحقة الكود

من الأول للآخر حتى نصل إلى المكان المطلوب فهذا يأخذ وقت وجهد كبير بعكس الدوال.

والدليل على أهميته الدوال في البرمجة نأخذ المثال التالي:

لو أردنا كتابة خطوات صنع كأس من الشاي فأننا نكتب ما يأتي:-

1-ضع الماء في غلاية الشاي
2-سخن الماء حتى يغلى
3-أضف شايا إلى الماء
4-أضف سكر إليه
5-أطفئ النار
6-صب شايا في الكأس.

افرض ألان أننا نعمل في مقهى ونود طلب كأس من الشاي في كل مره ، بدل من كتابه ال6 خطوات في كل مره
نقوم بكتابه ال6 خطوات في Function في أن الخطوات التي نحتاج كأساً من الشاي نقوم بكتابه خطوه واحده فقط وهي:
1-استدعاء كأس من الشاي.
تخيل الآن كم وفرنا من الخطوات والوقت في كتابة الكود وخاصة إذا كان البرنامج يتطلب حسابات وعمليات كثيرة وكم يكون البرنامج سهلاً واضحاً .

Functionتأخذ الشكل التالي:

كود:

```
Type-specified function-name (formal
parameters)
```

```
{
function body
}
```

أنواع:- Function

in *تأخذ , وتكون Pass by value :
in & out *تأخذ وترجع , وتكون Pass by reference :
out *ترجع , وتكون Output parameters :

وتكون أحد الأشكال التالية:-

1-تأخذ وترجع
2-لا تأخذ وترجع
3-تأخذ ولا ترجع
4-لا تأخذ ولا ترجع

ويتم الاتصال أو استدعاء function بالاسم وعلى حسب ما تأخذ بنفس النوع والعدد.

تنقسم الـ Functions إلي قسمين:-

1- Functionsجاهزة وتكون built in داخل برنامج C++ ويتم استدعاءها مباشرة ولكن يتطلب استدعاء Header File الخاص فيها مثال على ذالك:

كود:

```
cos(5);        //  5 للرقم cos هذه الدالة 
سوف ترجع الزاوية
log(5);        // 5 للرقم log رجع
pow(4,2);   //   وهذه الدالة أيضا سوف ترجع
2 أس 4 الرقم من الناتج
sqrt(4);        //   وهذه سوف ترجع جزر للرقم
4
```

وغيرها من الدوال الجاهزة ولكن الدوال هذه الخاصة بالرياضة يتطلب منك استدعاء Header File الخاص فيه وهو

كود:

```
#include<math.h>
```

2- Functions والتي يتم إنشاءها من قبل المبرمج لاستخدامها في البرنامج (سوف يتم التركيز على هذا الجزء) مثال على ذالك:

كود:

```
#include<iostream.h>
int cube(int x)
{
return x*x*x;
}
void main()
{
cube(2);
}
```

تم إنشاء دالة وقد تم تسميتها cube وعلى حسب ما ترغب وهي من نوع int

(سوف يتم شرح ذالك لاحقاً)
وهي تأخذ متغير واحد من نوع int وداخلها ترجع القيمة الناتجة من مضروب المتغير x في نفسه 3 مرات إي مكعب العدد.

وفي جزء main تم الاتصال في الدالة cube بالاسم فقط وتم إرسال القيمة 2 لأنها تأخذ متغير من النوع int لذا يجب إرسال القيمة من نفس النوع وبنفس العدد إي لو كانت تأخذ أكثر من قيمة يتم إرسالها وسوف أعطي مثال على ذالك.
بعد الأتصال في الدالة وإرسال القيمة لها تقوم الدالة بوضع القيمة المستقبلة في مكانها بين الأقواس إي في المتغير x وبعد ذالك تقوم بضرب القيمة وإرجاعها للمكان الذي قام بالاتصال في الدالة إي ال mainوبالتحديد في السطر ;cube(2) والذي تم إرجاع القيمة هو كلمة return وهذه الكلمة تقوم الخروج من الدالة كلياً والرجوع إلي المكان الذي تم استدعاها إي لو وضعت بعد كلمة return إي كود لن يتم تنفيذه ولن يمر عليه
وعندا انتهاء عمل الدالة والخروج منها تنتهي وتموت وتضل موجودة منتظره إي اتصال ثاني حتى تعمل.

لو أخذت الكود وعملت له run سوف تلاحظ لم يتم إخراج القيمة على الشاشة لماذا ؟؟؟

القيمة تم إرجاعها ولكن لم يتم إمساكها في متغير وحفظها أو إخراجها على الشاشة مباشرة لذا القيمة ضاعت

لكي يتم حفظ القيمة يتطلب منك تعريف متغير من نفس النوع الذي سوف يرجع من الدالة مثل

كود:

```
int c = cube(2);
```

وبعدها تطبع المتغير c ليظهر القيمة الآتي من الدالة أو بإمكانك طباعة القيمة مباشره دون تعريف متغير مثل:

كود:

```
cout<< cube(2);
```

مثال على دالة تأخذ أكثر من متغير:

كود:

```
#include<iostream.h>
float max(int x , float y)
{
if(x > y)
return x;
return y;
}
void main()
{
cout<<max(5,5.4);
}
```

لاحظ هذا المثال الدالة تأخذ متغيرين احدهم من نوع عدد صحيح إي int والثاني كسر إي float لذا تم إرسال قمتين وبنفس النوع وتكون القمتين بنفس ترتبها في الدالة إي الأول int والثاني float لذا الأول تم وضعه رقم صحيح والثاني كسر وهو سوف يأخذ كل قيمة ووضعها في الدالة بالترتيب.

هذه الدالة تقوم بإرجاع القيمة الكبيرة بين قمتين إذا قيمة x اكبر من قيمة y إذا return xإي اخرج من الدالة بقيمة x وإذا كان لأ إذا عدي الخطوة هذه وكمل الكود وهو return y لماذا لم يتم وضع
else return yوإن وضعتها لا يضر أبداً ولكن من العقل إن لم تكون x اكبر من yفأكيد y هي الأكبر
ولن يدخل إلي الخطوة الأخيرة إلا إذا لم تحقق الشرط في الخطوة التي قبلها وإن تحقق الشرط فلن يدخل إلى الخطوة الأخيرة بما فيه من كلمة return لذا يتم الاكتفاء من وضع كلمة return y بدون. else

وهكذا نكون قد علمنا إنه يجب عندا الاتصال بأي دالة يتم ذالك بحسب العدد ما تأخذه وبنفس النوع ويكون بالترتيب.

وهذا المثالين هو من النوع الأول من أشكال الـ Function إي أنها) تأخذ وترجع (إي أخذت قيمة أو أكثر وترجع وهو return بأنها تقوم بترجيع قيمة إلي المكان الذي اتصل بها

والآن سوف نأخذ الأشكال التالية ولكل واحد مثال

ولكن قبل الدخول في الأشكال الثانية سوف اشرح الجزء من الذي لم اشرحه من الدالة وهو مهم جداً عندا تعريف الدوال.

أكيد لاحظت في المثال الأول تم تعريف الدالة cube من نوع int والدالة max من نوع float ولكن هل تعرف لماذا وماذا تعني.... ؟؟؟؟

إذا كانت الدالة ترجع قيمة إي return فيجب يتم تعرفيها من نفس النوع الذي سوف يتم إرجاعه في المثال الأول كانت القيمة التي سوق يتم إرجاعها هي مكعب الرقم 2 والناتج سوف يكون 8 وهو عدد صحيح لذا تم تعرفيها من نوع. int و الدالة الثانية سوف يكون الراجع إما عدد صحيح أو كسر على حسب الأكبر لذا تم وضعها float لأنه float يحمل الرقم الصحيح والكسر بينما int يحمل رقم صحيح فقط.

لذا يجب تعرف الدالة من نفس النوع الذي سوف تتم إرجاعه وهذا إذا كانت ترجع بعض النظر إذا كانت تأخذ أم لا ، إما إذا كانت لا ترجع شياً فيتم وضع كلمة void وسوف يتم شرح هذا النوع.

الآن ندخل إلي الشكل الثاني وهو) لا تأخذ وترجع(

لاحظ المثال التالي :-

كود:

```
#include<iostream.h>
int sum()
{
int x = 5 , y = 4;
return x + y;
}
void main()
{
int z = sum();
cout<<z;
}
```

في هذا المثال الدالة لا تأخذ شيء لذا الأقواس فارغة ولكن تقوم بإرجاع قيمة صحيحة لذا تم تعريفها من نوع int للدالة ، ولاحظ عندا الاتصال في الدالة تم استدعائها بالاسم ولم يتم إرسال إي قيمة ولكن بما إنها تقوم بإرجاع قيمة فتم تعريف متغير لكي نقوم بحفظ القيمة الراجعة من الدالة لذا تم تعرف متغير بنفس النوع أو بإمكانك طباعة القيمة مباشرة إذا كنت لا ترغب بحفظها كما الأمثلة السابقة.

كود:

```
cout<<sum();
```

الآن نأتي القسم الآخر و إلي الشكل الثالث وهو) تأخذ ولا ترجع(

مثال علي ذالك:-

كود:

```
#include<iostream.h>
void cube(int x)
{
cout<<x*x*x;
}
void main()
{
cube(2);
}
```

لاحظ هذا المثال وهو نفس المثال الأول ولكن الدالة هذه لا يوجد قيمة يتم إرجاعها returnولذا تم تعريف الدالة من نوع void ولكن سوف تخرج القيمة إلي الشاشة مباشرة ولن يتم إرجاع إي شيء للمكان الذي قام باتصال فيها ولذا تم استدعاها بالاسم فقط وبما إنها تأخذ قيمة فتم إرسال القيمة وبنفس النوع ولم يتم وضع

كود:

```
cout<<cube(2);
// OR
int z =    cube(2);
```

والآن الشكل الأخير وهو) لا تأخذ ولا ترجع(

كود:

```
#include<iostream.h>
void sum()
{
int x = 5 , y = 4;
int z = x + y;
cout<<z;
}
void main()
{
sum();
}
```

لاحظ هنا الدالة من نوع void إي لا ترجع شيء وبنفس الوقت الأقواس فارغة إي لا تأخذ أيضا.

معلومة مهمة جداً جداً : الدوال لا تحفظ الذي تقوم بعمله ولا تغير في القيم التي يتم إرسالها إليها
كيف ولماذا وكيف نقوم بحفظ التغيرات؟؟؟

مثال لكي يتم توضح الصورة:-

كود:

```
#include<iostream.h>
void element(int x)
{
x++;
cout<<x<<endl;
}
```

```
void main()
{
int x = 5;
element(x);
cout<<x<<endl;
}
```

الناتج على الشكل التالي:

كود:

```
6
5
```

تم تعريف في الجزء main متغير ويحمل قيمة 5 وتم إرسالها إلي الدالة والدالة قامت باستقبالها ووضعها في متغير آخر يدعى x ولكن يختلف عن x الذي في mainحيث يتم إرسال القيمة التي بداخل المتغير x وليس x نفسه وداخل الدالة قمنا بزيادة المتغير x واحد وعندما قمنا بطباعة في الدالة ظهر لنا 6 إي6=5+1 ولكن في جزء main وبعدا الاتصال في الدالة عندما قمنا بطباعة المتغير x فقط طبع الرقم 5 وكما هي إي لم يقوم بزيادة واحد على المتغير.

ولكن كيف يتم حفظ المتغير أو إرجاع ما تم تغيره كما المثال التالي:-

كود:

```
#include<iostream.h>
int element(int x)
{
x++;
cout<<x<<endl;
return x;
}
void main()
{
```

```
int x = 5;
x = element(x);
cout<<x<<endl;
}
```

الناتج على الشكل التالي:

كود:

```
6
6
```

هنا لم يتم التغير في القيمة نفسها في الدالة ولكن تم إرسال القيمة التي في المتغير وبعد التعديل تم إرجاعها إلي **main** وتم وضع القيمة بعد التغير في **x** نفسه مما تم إرجاع القيمة وحفظها مره أخرى بنفس المتغير ولكن هذا يتطلب منك أن تكون الدالة ترجع وعلى هذا الشكل.

ومعلومة أخرى الدالة لا تستطيع إرجاع أكثر من قيمة واحدة فقط انظر هذا المثال:

كود:

```
#include<iostream.h>
int element(int x, int y)
{
x++;
y--;
cout<<x<<"  "<<y<<endl;
return x;
return y;
}
void main()
{
int x = 5 ,y = 4;
```

```
x = element(x,y);
cout<<x<<"  "<<y<<endl;
}
```

الناتج:

كود:

```
6   3
6   4
```

هنا تم إرسال قمتين والتغير فيهم وقمنا بوضع return لكل متغير ولكن كما نعلم إنه عند الدخول إلي كلمة return يتم الخروج وموت الدالة ولن يمر بالسطر الآخر وعندا استقبالها يتم وضعها في متغير واحد لاستطيع وضع أكثر من متغير لأكثر من قيمة

ولكن هل توجد طريقة يتم حفظ التغير في المتغير نفسه وبدون إرجاع القيمة ووضعها مره أخرى في المتغير نفسه وهل توجد طريقة لحفظ أكثر من متغير .....؟؟؟

نعم يوجد طريقة وتدعى:: Reference Passing BY
بهذه الطريقة يتم إرسال القيم ويتم استقبالهم كا Reference إي عنوان القيمة في الذاكرة الجهاز مما يؤدي إلي الذهاب إلي المتغير نفسه عن طريق عنوانه في الذاكرة و التغيّر فيه ، كل الذي يتطلب لعمل ذالك وضع علامة & إمام تعريف المتغير داخل الأقواس في الدالة مثال:-

كود:

```
#include<iostream.h>
void element(int& x, int& y)
{
```

```
x++;
y--;
cout<<x<<"   "<<y<<endl;
}
void main()
{
int x = 5 ,y = 4;
element(x,y);
cout<<x<<"   "<<y<<endl;
}
```

والناتج على الشكل التالي:-

كود:

```
6   3
6   3
```

لاحظ الناتج عندا طباعته في الدالة وبعد التغير في القيم وطباعة في جزء main كان الناتج نفسه ولم يتم وضع إي return لإرجاع القيمة التي تغيرت إذا فهو قام التغير في الدالة نفسها وكل الذي تطلب وضع العلامة & في المتغيرات التي نريدها لحفظ التغير.

مثال آخر وعلي نفس الجزء:-

كود:

```
#include<iostream.h>
void change(int x, int& y)
{
x = 55;
y = 44;
cout<<x<<"   "<<y<<endl;
}
void main()
{
```

```
int x = 5 ,y = 4;
cout<<x<<"  "<<y<<endl;
change(x,y);
cout<<x<<"  "<<y<<endl;
}
```

الناتج:-

كود:

```
5   4
55   44
5    44
```

وكما لاحظت لم يتم وضع علامة & على المتغير x لذا إي تعديل أو تغير لن يؤثر في المتغير الأصلي وهذا النوع يدعى.(Passing By Value)
لكن تم وضعها على المتغير y لذا إي تعديل سوف يؤثر فيها وهذا يدعى (Passing By Reference).

والآن بعدا ما عرفنا أقسام وأنواع وإشكال الـ Function ندخل على جزء ويدعى Overloading

وهو أكثر من دالة لها نفس الاسم ولكن يختفوا من حيث إلى تأخذه و عددهم و نوعهم مثال على ذالك:-

كود:

```
#include<iostream.h>
int sum(int x, int y)
{
```

```
return x + y;
}
double sum(double x, int y)
{
return x + y;
}
int sum(int x, int y , int z)
{
return x + y + z;
}
void main()
{
cout<<sum(5,4)<<endl;
cout<<sum(5.1,4)<<endl;
cout<<sum(5,4,1)<<endl;
}
```

لاحظ هناك 3 دوال لهم نفس الاسم ولكن يختلفوا فما يستقبلوا من متغيرات الأولي تأخذ متغيرين من نوع **int** والثانية متغيرين أيضاَ ولكن احد المتغيرات من نوع **double**لذا تكون تختلف عن الأولي ، والثالثة تأخذ 3 متغيرات لذا كل دالة تختلف عن الأخرى ولكل وحده يكون عمل مختلف ولكن مجرد تشابه أسماء

وعندا الاتصال بدالة معينه يتم معرفة ذالك آلياَ من قبل المترجم بحسب ما تم إرساله للدالة و بان يكون بنفس الدالة المراد الاتصال بها.

طريقة تعريف:- Function

طريقين:-

1- Definition
2- Declaration

Definition : إي تكون الدالة فوق الذي سوف تقوم بالاتصال فيها مثل

```
#include<iostream.h>

int cube(int x)
{
    return x*x*x;
}

void main()
{
            cube(2);
}
```

تم الاتصال من الأسفل إلي الأعلى

وكل الأمثلة التي طرحت كانت من هذا النوع من طريقة التعريف

Declaration : وتدعي أيضاً Header وتكون الدالة أسفل الذي سوف يقوم بالاتصال فيها مثال

```
#include<iostream.h>

void main()
{
      cube(2);
}

int cube(int x)
{
      return x*x*x;
}
```

يتم الاتصال من الأعلى إلي الأسفل

هنا يكون الاتصال من الأعلى إلي الأسفل وكلا الطريقتين صحيحة و بإمكانك استخدام إي طريقة في تعريف الدوال.
ولكن في النوع Declaration إن كتب الكود الذي في المثال مثل ما هو سوف يعطي error و يقول بأنه cube غير معرفه مع العلم إنها موجودة ولكنها تعتبر Headerلاستطيع إن يراها وهي بالأسف لذا أطلق على هذا النوع الاسم Headerأيضا لكي تقدر تتعامل مع هذا النوع عليك تعريفها في الأعلى في فوق كل الدوال وكأنها Header File بكتابة السطر

كود:

```
cube(int);
```

إي تكتب اسم الدالة ولا يكتب نوعها إذا كانت ترجع أم لا
ويكتب أيضا إذا كانت تأخذ ولكل واحد نوعه فوق ولا يكتب اسم المتغيرات التي تأخذها وفي الآخر فاصلة منقوطة ; إي لا يضع. {}

إذا يصبح المثال على الشكل التالي:

كود:

```
#include<iostream.h>
cube(int);
```

```
void main()
{
cube(2);
}
int cube(int x)
{
return x*x*x;
}
```

مثال إذا كانت الدالة تأخذ متغيران من الأول نوع float والثاني نوع int وهو Referenceوالدالة من نوع void إي لن ترجع شيء فيكتب في الأعلى

كود:

```
change(float ,int&);
```

وليس شرط من main التي تتصل في الدالة ممكن تكون دالة تتصل بدالة أخرى غير main ولكن يكون الانطلاق منmain

مثال علي ذالك:-

كود:

```
#include<iostream.h>
cube(int);
void call()
{
cube(2);
}
```

```
void main()
{
call();
}
int cube(int x)
{
return x*x*x;
}
```

لاحظ هذا المثال الدالة main قامت بالاتصال بالدالة call والدالة call قامت بالاتصال في الدالة cube وأرسلت لها القيمة ولكن الانطلاق كان من الدالة main ولكن الدالة call اتصلت في الدالة cube من الأعلى إلي الأسفل إي بطريقة Declarationلذا عرفنا الدالة cube في الأعلى بأنها.Header

وكما قلنا سابقاً الـ main تدعي دالة (Function) وعملها مثل عمل إي دالة يتم الاتصال فيها عن طريق المترجم أول ما يعمل البرنامج فإنه يذهب إلي الدالة mainآلياً وينتهي عملها وتموت عندا نهاية البرنامج) عندما يقفل البرنامج. (

أتمنى أن يكون الشرح واضحاً وأن أكون غطيت أكبر جزء في Functions (الدوال.)